Intel® Integrated Performance Primitives for Intel® Architecture

Implementation of Intel® Image Processing Library by Intel® Integrated Performance Primitives

Version 3.0

June, 2004

Table of Contents

Introduction

The document briefly describes the Intel® Image Processing Library (IPL) implemented using Intel® Integrated Performance Primitives (Intel® IPP) and also provides source code examples that may serve as illustrations of how you can use Intel IPP in image processing applications. The Intel IPP software is a new generation of the Intel® Performance Libraries, comprising a broad range of functions for basic software functionality and including, among many others, the image processing functions subset.
The Intel IPP package itself can be used instead of IPL. Among the reasons why Intel IPP is preferable over IPL are:

- Intel IPP is faster on Intel® Pentium® 4 processor;
- Intel IPP does not contain complicated data structures (such as `IplImage` in IPL);
- Intel IPP is optimized for Intel® Itanium® processor family;
- Intel IPP is also available for the Linux* operating system.

You can also use the Intel IPP version of the Intel Image Processing Library that was specially designed to improve performance of existing IPL applications and help developers to migrate to Intel IPP. The Intel IPP version of IPL gives an opportunity to enhance performance and effectiveness of existing applications without having to modify the source code.

Note that there are two limitations to Intel IPP use in image processing, particularly as compared to the older Intel Image Processing Library:

- Intel IPP does not support the `1u` data type;
- Intel IPP does not support direct processing of tiled images.

The first limitation is not usually a problem because 1-bit images are not widely used.

To remove the second constraint, code should be provided that enables Intel IPP functions to process an image in the tile-by-tile mode. A separate sample demonstrates how tiling support can be accomplished using Intel IPP.

Additional information on Intel IPP as well as other Intel® software performance products is available at http://www.intel.com/software/products/.

To give feedback or report any problems with installation or use of the Intel IPP software, please contact via Intel® Premier Support at https://premier.intel.com. For registration and self help information, please check http://support.intel.com/support/performancetools/libraries/ipp.

Error Handling

One of the main differences between IPL and Intel IPP is the handling of errors. When an IPL function detects an error (for example, the null pointer is passed to the function), the function sets a special variable in the library, whereas Intel IPP functions return the status code directly. This status code may be analyzed and the correspondent string with a brief status description can be displayed. For example:

```
IppiSize sz = { -100, 0 };
IppStatus st = ippiSet_32f_C1R( 255, buf, wstep, sz );
if( ippStsNoErr != st )
   printf( "--err(%d) %s\n", st, ippGetStatusString( st ) );
```

The following error message will be displayed to indicate that the buffer size has an illegal value:

```
--err(-6) Wrong value of data size
```

It is important to know that, in contrast to IPL, the Intel IPP functions can distinguish fatal errors (when the operation is aborted) from the problems that allow to complete the operation, for example, when an overflow occurs.

Special cases

Since not every mode of each Image Processing Library function has a corresponding function in the Intel IPP package, implementing some of the functions may prove an intricate task. Some special cases of function implementation are considered below.

COI Processing

To accomplish processing of a single channel (COI) of a multi-channel pixel image, several stages are needed, which include:

- copying the particular channel;
- separate processing of this channel;
- copying the processed data back to the image.

The following is the exemplary code segment that processes only one channel in a 3-channel image:

```
ippiCopy_8u_C3C1R( src, tmp );
// processing by func 8u_C1R( tmp );
ippiCopy_8u_C1C3R( tmp, dst );
```

Planar Image Processing

In the absence of a corresponding Intel IPP primitive with P3 or P4 descriptor, multi-channel planar images are processed in the loop as given by:

```
for( i=0; i<channels; i++ ) {
   func_C1R;
}
```

Operations on Images with Mask

Only two types of Intel IPP image processing primitives process images with mask, namely `ippiSet` and `ippiCopy`. Therefore, the implementation of the IPL functions in the mask mode requires calls to several primitives to prepare an output mask, process the whole image, and then copy the processed data to the destination with the mask.

The code segment below illustrates this approach:

```
ippiSet_8u_C1R(255, allMask);      // common mask
ippiAnd_8u_C1IR(mask, allMask);    // And with every mask src and dst
func_C?R(tmp);
ippiCopy_C?MR(tmp, dst, allMask);
```

Functions Requiring Border Pixels

In the current Intel IPP implementation, the IPL functions that require border pixels to compute pixel values at the edge of an image (for example, filtering functions) create a temporary image larger than the original one. Depending on the border mode, these functions fill the border of that image with the corresponding pixels using the following primitives:

`ippiSet` - for `IPL_BORDER_CONSTANT` mode;
`ippiCopy, ippiSet` - for `IPL_BORDER_REPLICATE` mode;
`ippiMirror` - for `IPL_BORDER_REFLECT` mode;
`ippiCopy` - for `IPL_BORDER_WRAP` mode.

Functions Map

Table 1 gives the complete list of IPL functions together with the corresponding Intel IPP primitives used in the implementation. The rightmost column of the table presents the average speedup values that show how much faster the Intel IPP version of the particular IPL function is with respect to the original IPL function. The measurements were done on the Intel Pentium 4 processor-based system. It must be noted that several original IPL functions showed a faster performance than the Intel IPP analogs (speedup is less than 1).

Table 1. Functions Map

IPL Function	Intel IPP function	Speedup
iplAbs (8U, 16U, 16S, 32F)	ippiAbs (16s, 32f); ippiCopy (8u, 16u)	2.8
iplAdd (8U, 16S, 32S, 32F)	ippiAdd (8u, 16s, 32f); ippsAdd (32s)	1.0
iplAddS (8U, 16S, 32S)	ippiAddC, ippiSubC (8u, 16s); ippsAddC (32s)	1.5
iplAddSFP (32F)	ippiAddC_32f	1.0
iplAlphaComposite (8U, 16U)	ippiCopy, ippiAlphaComp, ippiConvert, ippiDiv	1.9
iplAlphaCompositeC (8U, 16U)	ippiAlphaCompC, ippiMulC	6.5
iplAnd (8U, 8S, 16U, 16S, 32S)	ippiAnd (8u, 16u, 32s)	1.5
iplAndS (8U, 8S, 16U, 16S, 32S)	ippiAndC (8u, 16u, 32s)	1.3
iplApplyColorTwist (8U, 16U)	ippiColorTwist	0.7
iplBitonalToGray (8U, 16S)	ippiThreshold_LTVal, ippiThreshold_GTVal, ippiThreshold_LTValGTVal	
iplBlur (8U, 16S, 32F)	Image with border creation, ippiCopy, ippiFilterBox	1.4
iplCcsFft2D (32S8U, 32F8U) FFT	ippiFFTInitAlloc_R, ippiFFTGetBufSize_R, ippiFFTInv_PackToR, ippiFFTFree_R,	0.9
iplCcsFft2D (32S8U, 32F8U) DFT	ippiDFTInitAlloc_R, ippiDFTGetBufSize_R, ippiDFTInv_PackToR, ippiDFTFree_R, ippiConvert_32f8u	0.9
iplCentralMoment (8U, 32F)	ippiMomentInitAlloc_64f, ippiMoments64f, ippiGetCentralMoment_64f, ippiMomentFree_64f	
iplClose (8U)	Image with border creation, ippiCopy, ippiDilate3x3_8u, ippiErode3x3_8u. See example	2.5
iplColorToGray (8U, 16U)	ippiColorToGray, ippiScale, ippiCopy_P3C3R for plane	0.9
iplColorMedianFilter (8U, 16S)	Image with border creation, ippiCopy, ippiFilterMedianColor	2.4
iplColorTwistFP (32F)	ippiColorTwist	2.5
iplComputeHisto (8U)	ippiHistogramRange_8u	1.0
iplContrastStretch (8U)	ippiLUT_8u	0.8
iplConvert (8u -> 8u)	ippiCopy_8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (8u -> 8s)	ippiThreshold_GTVal_8u[, ippiCopy	

	C<>P, ippiMirror]	
iplConvert (8u -> 16u)	ippiConvert_8u16u[, ippiCopy C<>P, ippiMirror]	
iplConvert (8u -> 16s)	ippiConvert_8u16s[, ippiCopy C<>P, ippiMirror]	
iplConvert (8u -> 32s)	ippiConvert_8u32s[, ippiCopy C<>P, ippiMirror]	
iplConvert (8s -> 8u)	ippiThreshold_GTVal_8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (8s -> 8s)	ippiCopy_8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (8s -> 16u)	ippiConvert_8u16u, ippiThreshold_GTVal_16s[, ippiCopy C<>P, ippiMirror]	
iplConvert (8s -> 32s)	ippiConvert_8s32s[, ippiCopy C<>P, ippiMirror]	
iplConvert (16u -> 8u)	ippiConvert_16u8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (16u -> 8s)	ippiConvert_16u8u, ippiThreshold_GTVal_8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (16u -> 16u)	ippiCopy_16s[, ippiCopy C<>P, ippiMirror]	
iplConvert (16u -> 16s)	ippiThreshold_LTVal_16s[, ippiCopy C<>P, ippiMirror]	
iplConvert (16s -> 8u)	ippiConvert_16s8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (16s -> 16u)	ippiThreshold_LTVal_16s[, ippiCopy C<>P, ippiMirror]	
iplConvert (16s -> 16s)	ippiCopy_16s[, ippiCopy C<>P, ippiMirror]	
iplConvert (32s -> 8u)	ippiConvert_32s8u[, ippiCopy C<>P, ippiMirror]	
iplConvert (32s -> 8s)	ippiConvert_32s8s[, ippiCopy C<>P, ippiMirror]	
iplConvert (32s -> 32s)	ippiCopy_32f[, ippiCopy C<>P, ippiMirror]	
iplConvolve2D (8U, 8S, 16U, 16S, 32S), one kernel	Image with border creation, ippiCopy, ippiFilter (8u, 16s)	1.4
iplConvolve2D (8U, 8S, 16U, 16S, 32S), several kernels	Image with border creation, ippiCopy, ippiConvert_??32f, loop by kernels: ippsConvert_32s32f, ippiDivC_32f_C?IR *или* ippiMulC_32f_C?IR, ippiSqr_32f_C?IR[, ippsMinEvery_32f, ippsMaxEvery_32f, ippiAdd_32f_C?IR, ippiSqrt_32f_C?IR], ippiConvert_32f??	
iplConvolveSep2D (8U, 16S)	Image with border creation, ippiCopy, ippiConvert_?32f, ippiConvert_32f?, ippsConvert_32s32f, ippiFilterColumn, ippiFilterRow	0.6
iplConvolveSep2DFP, (32F)	Image with border creation, ippiCopy, ippiFilterColumn_32f, ippiFilterRow_32f	2.3
iplConvolve2DFP, (32F)	Image with border creation, ippiCopy_32f, ippiFilter_32f, ippiSqr_32f, ippsMinEvery_32f, ippsMaxEvery_32f, ippiAdd_32f, ippiSqrt_32f	1.3

iplCopy (8U, 8S, 16U, 16S, 32S, 32F)	ippiCopy, ippiCopy_MR	1.9
iplCreateImageJaehne (8U, 8S, 16U, 16S, 32S, 32F)	ippiImageJaehne	
iplDCT2D (8U16S, 8U32F)	ippiDCTFwdInitAlloc_32f, ippiDCTFwdGetBufSize_32f, ippiConvert_8u32f, ippiDCTFwd_32f_C3R[, ippiConvert_32f16s], ippiDCTFwdFree_32f [ippiDCT8x8FwdLS_8u16s_C1R]	
iplDCT2D (16S8U, 32F8U)	ippiDCTInvInitAlloc_32f, ippiDCTInvGetBufSize_32f[, ippiConvert_16s32f], ippiDCTInv_32f, ippiConvert_32f8u, ippiDCTInvFree_32f [ippiDCT8x8InvLSClip_16s8u_C1R]	
iplDecimate (8U, 16U, 32F)	ippiResize	3.2
iplDecimateBlur (8U, 32F)	Image with border creation, ippiCopy, ippiFilterBox, ippiResize	0.6
iplDilate (8U)	Image with border creation, loop ippiDilate3x3_8u	3.1
iplEqual (8U, 16S, 32F)	ippiCompare [ippiAnd_8u_C1IR]	1.5
iplEqualS (8U, 16S)	ippiCompareC [ippiAnd_8u_C1IR]	1.3
iplEqualSFP	ippiCompareC_32f [ippiAnd_8u_C1IR]	1.1
iplEqualSFPEps	ippiCompareEqualEpsC_32f [ippiAnd_8u_C1IR]	1.2
iplEqualFPEps	ippiCompareEqualEps_32f [ippiAnd_8u_C1IR]	1.5
iplErode (8U)	Image with border creation, ippiCopy, loop ippiErode3x3_8u	3.1
iplExchange (8U, 8S, 16U, 16S, 32S, 32F)	ippiCopy, ippiCopy_MR	1.4
iplFixedFilter (8U, 16S, 32F)	Image with border creation, ippiCopy, ippiFilterPrewittVert, ippiFilterPrewittHoriz, ippiFilterSobelVert, ippiFilterSobelHoriz, ippiFilterLaplace, ippiFilterGauss, ippiFilterHipass, ippiFilterSharpen	1.8
iplGetAffineTransform	ippiGetAffineTransform	
iplGetAffineTransformROI	ippiGetAffineTransform	
iplGetAffineQuad	ippiGetAffineQuad	
iplGetAffineQuadROI	ippiGetAffineQuad	
iplGetAffineBound	ippiGetAffineBound	
iplGetAffineBoundROI	ippiGetAffineBound	
iplGetBilinearQuad	ippiGetBilinearQuad	
iplGetBilinearQuadROI	ippiGetBilinearQuad	
iplGetBilinearBound	ippiGetBilinearBound	
iplGetBilinearBoundROI	ippiGetBilinearBound	
iplGetBilinearTransform	ippiGetBilinearTransform	
iplGetBilinearTransformROI	ippiGetBilinearTransform	
iplGetPerspectiveQuad	ippiGetPerspectiveQuad	
iplGetPerspectiveQuadROI	ippiGetPerspectiveQuad	
iplGetPerspectiveBound	ippiGetPerspectiveBound	
iplGetPerspectiveBoundROI	ippiGetPerspectiveBound	
iplGetPerspectiveTransform	ippiGetPerspectiveTransform	
iplGetPerspectiveTransformROI	ippiGetPerspectiveTransform	
iplGetPixel (8U, 8S, 16U, 16S, 32S,	ippiCopy	0.8

32F)		
iplGetRotateShift	ippiAddRotateShift	
iplGrayToColor (8U) pixel	ippiCopy_8u_C1C3R(C1C4R), ippiMulC_8u_C3IR(AC4R)	1.7
iplGrayToColor (8U) plane	ippiMulC_8u_C1R	
iplGreater (8U, 16S, 32F)	ippiCompare [ippiAnd_8u_C1IR]	1.2
iplGreaterS (8U, 16S)	ippiCompareC [ippiAnd_8u_C1IR]	1.3
iplGreaterSFP	ippiCompareC_32f [ippiAnd_8u_C1IR]	1.0
iplHistoEqualize (8U)	ippiLUT_8u, ippiHistogramRange_8u	0.7
iplHLS2RGB (8U, 16U, 32F) pixel	ippiHLSToRGB_C3R(AC4R) ippiSwapChannels	1.9
iplHLS2RGB (8U, 16U, 32F) plane	ippiCopy_P3C3R ippiHLSToRGB_C3R ippiCopy_C3P3R	
iplHSV2RGB (8U, 16U) pixel	ippiHSVToRGB_C3R(AC4R) ippiSwapChannels	2.5
iplHSV2RGB (8U, 16U) plane	ippiCopy_P3C3R ippiHSVToRGB_C3R ippiCopy_C3P3R	
iplLess (8U, 16S, 32F)	ippiCompare ippiAnd_8u_C1IR	1.2
iplLessS (8U, 16S)	ippiCompareC ippiAnd_8u_C1IR	1.3
iplLessSFP	ippiCompareC_32f ippiAnd_8u_C1IR	1.0
iplLShiftS (8U, 8S, 16U, 16S, 32S)	ippiLShiftC	1.5
iplLUV2RGB (8U, 16U, 32F) pixel	ippiLUVToRGB_C3R(AC4R) ippiSwapChannels	0.6
iplLUV2RGB (8U, 16U, 32F) plane	ippiCopy_P3C3R ippiLUVToRGB_C3R ippiCopy_C3P3R	
iplMaxFilter (8U, 16S)	Image with border creation, ippiCopy ippiFilterMax	2.0
iplMinFilter (8U, 16S)	Image with border creation, ippiCopy ippiFilterMin	2.1
iplMedianFilter (8U, 16S)	Image with border creation, ippiCopy ippiFilterMedian	2.0
iplMinMaxFP (32F)	ippiMinMax_32f	2.0
iplMirror (8U, 8S, 16U, 16S, 32S, 32F)	ippiMirror	1.3
iplMoments (8U, 32F)	ippiMomentInitAlloc_64f, ippiMoments64f, ippiGetSpatialMoment_64f, ippiGetCentralMoment_64f, ippiMomentFree_64f	0.4
iplMpyRCPack2D (8S, 16S, 32F)	ippiConvert ippiMulPack	0.5
iplMultiply (8U, 16S, 32S, 32F)	ippiMul	1.0
iplMultiplyS (8U, 16S, 32S)	ippiMulC	1.2
iplMultiplySFP (32F)	ippiMulC	0.9
iplMultiplyScale (8U, 16U)	ippiMulScale	2.4
iplMultiplySScale (8U, 16U)	ippiMulCScale	2.4
iplNoiseImage (8U, 16S, 32F)	ippiAddRandUniform_Direct ippiAddRandGauss_Direct	0.8
iplNorm (8U, 16S, 32F)	ippiNorm_Inf, ippiNorm_L1, ippiNorm_L2, ippiNormDiff_Inf, ippiNormDiff_L1, ippiNormDiff_L2, ippiNormRel_Inf, ippiNormRel_L1, ippiNormRel_L2	0.6
iplNormalizedSpatialMoment (8U, 32F)	ippiMomentInitAlloc_64f, ippiMoments64f, ippiGetNormalizedSpatialMoment_64f, ippiMomentFree_64f	
iplNormalizedCentralMoment (8U, 32F)	ippiMomentInitAlloc_64f, ippiMoments64f, ippiGetNormalizedCentralMoment_64f,	

	ippiMomentFree_64f	
iplNormCrossCorr (8U, 8S, 16U, 16S, 32S, 32F)	ippiCrossCorrSame_Norm, ippiConvert, ippiMulC, ippsConvert_32s32f, ippsConvert_32f32s	8.8
iplNot (8U, 8S, 16U, 16S, 32S)	ippiXorC	1.3
iplOpen (8U)	Image with border creation, ippiCopy ippiErode3x3_8u ippiDilate3x3_8u	2.6
iplOr (8U, 8S, 16U, 16S, 32S)	ippiOr	1.5
iplOrS (8U, 8S, 16U, 16S, 32S)	ippiOrC	1.3
iplPreMultiplyAlpha (8U, 16U)	ippiAlphaPremul, ippiSet, ippiAlphaPremulC, ippiMulScale	2.3
iplPutPixel (8U, 8S, 16U, 16S, 32S, 32F	ippiSet	0.7
iplRealFft2D (8U32S, 8U32F) FFT	ippiFFTInitAlloc_R, ippiFFTGetBufSize_R, ippiFFTFwd_RtoPack, ippiFFTFree_R_32s	1.5
iplRealFft2D (8U32S, 8U32F) DFT	ippiDFTInitAlloc_R, ippiDFTGetBufSize_R, ippiDFTFwd_RtoPack, ippiDFTFree_R ippiConvert_8u32f	
iplReduceBits (8U, 16U)	ippiReduceBits	1.6
iplRemap (8U, 32F)	ippiRemap	1.3
iplResize (8U, 16U, 32F)	ippiResize	1.0
iplRGB2HLS (8U, 16U, 32F) pixel	ippiSwapChannels ippiRGBToHLS_C3R(AC4R)	1.9
iplRGB2HLS (8U, 16U, 32F) plane	ippiCopy_P3C3R, ippiRGBToHLS_C3R, ippiCopy_C3P3R	
iplRGB2HSV (8U, 16U) pixel	ippiSwapChannels ippiRGBToHSV_C3R(AC4R)	2.1
iplRGB2HSV (8U, 16U) plane	ippiCopy_P3C3R, ippiRGBToHSV_C3R, ippiCopy_C3P3R	
iplRGB2LUV (8U, 16U, 32F) pixel	ippiSwapChannels ippiRGBToLUV_C3R(AC4R)	1.0
iplRGB2LUV (8U, 16U, 32F) plane	ippiCopy_P3C3R, ippiRGBToLUV_C3R, ippiCopy_C3P3R	
iplRGB2XYZ (8U, 16U, 32F) pixel	ippiSwapChannels ippiRGBToXYZ_C3R(AC4R)	1.6
iplRGB2XYZ (8U, 16U, 32F) plane	ippiCopy_P3C3R, ippiRGBToXYZ_C3R, ippiCopy_C3P3R	
iplRGB2YcrCb (8U) pixel	ippiSwapChannels ippiRGBToYCbCr_8u_C3R(AC4R)	2.0
iplRGB2YcrCb (8U) plane	ippiRGBToYCbCr_8u_P3R	
iplRGB2YUV (8U) pixel	ippiSwapChannels ippiRGBToYUV_8u_C3R(AC4R)	3.8
iplRGB2YUV (8U) plane	ippiRGBToYUV_8u_P3R	
iplRotate (8U, 32F)	ippiRotate	1.4
iplRShiftS (8U, 8S, 16U, 16S, 32S)	ippiRShiftC	1.4

iplScale (8u -> 8s)	ippiXorC_8u	1.1
iplScale (8u -> 16u)	ippiScale_8u16u	
iplScale (8u -> 16s)	ippiScale_8u16s	
iplScale (8u -> 32s)	ippiScale_8u32s	
iplScale (8s -> 8u)	ippiXorC_8u	
iplScale (8s -> 16u)	ippiXorC_8u, ippiScale_8u16u	
iplScale (8s -> 16s)	ippiXorC_8u, ippiScale_8u16s	
iplScale (8s -> 32s)	ippiXorC_8u, ippiScale_8u32s	
iplScale (16u -> 8u)	ippiScale_16u8u	
iplScale (16u -> 8s)	ippiScale_16u8u, ippiXorC_8u	
iplScale (16u -> 16s)	ippiXorC_16u	
iplScale (16s -> 8u)	ippiScale_16s8u	
iplScale (16s -> 8s)	ippiScale_16s8u, ippiXorC_8u	
iplScale (16s -> 16u)	ippiXorC_16u	
iplScale (32s -> 8u)	ippiScale_32s8u	
iplScale (32s -> 8s)	ippiScale_32s8u, ippiXorC_8u	
iplScaleFP (8u -> 32f)	ippiScale_8u32f	2.2
iplScaleFP (8s -> 32f)	ippiXorC_8u, ippiScale_8u32f	
iplScaleFP (32f -> 8u)	ippiScale_32f8u	
iplScaleFP (32f -> 8s)	ippiScale_32f8u, ippiXorC_8u	
iplSet (8U, 8S, 16U, 16S, 32S)	ippiSet, ippiSet_MR	2.0
iplSetFP 32F)	ippiSet, ippiSet_MR	1.2
iplShear (8U, 32F)	ippiShear	1.4
iplSpatialMoment (8U, 32F)	ippiMomentInitAlloc_64f, ippiMoments64f, ippiGetSpatialMoment_64f, ippiMomentFree_64f	
iplSquare (8U, 16U, 16S, 32F)	ippiSqr	1.1
iplSubtract (8U, 16S, 32S, 32F)	ippiSub ippsSub_32s	1.0
iplSubtractS (8U, 16S, 32S)	ippiSubC, ippiSet, ippiConvert, ippiMulC, ippiAddC, ippsSubC_32s, ippsSubCRev_32s	1.3
iplSubtractSFP (32F)	ippiSubC_32f ippsSubCRev_32f	1.0
iplThreshold (8U, 16S)	ippiSet, ippiThreshold_LTVal, ippiThreshold_GTVal, ippiScale	
iplWarpAffine (8U, 32F)	ippiWarpAffine	1.4
iplWarpBilinear (8U, 32F)	ippiWarpBilinear, ippiWarpBilinearBack	1.4
iplWarpBilinearQ (8U, 32F)	ippiWarpBilinearQuad, ippiGetBilinearTransform, ippiWarpBilinearBack	1.3
iplWarpPerspective (8U, 32F)	ippiWarpPerspective, ippiWarpPerspectiveBack	1.3
iplWarpPerspectiveQ (8U, 32F)	ippiWarpPerspectiveQuad ippiGetPerspectiveTransform ippiWarpPerspectiveBack	1.5
iplWtInit, iplWtInitUserFilter4, iplWtInitUserFilter, iplWtInitUserTaps	ippiWTFwdInitAlloc_32f_C1R, ippiWTFwdGetBufSize_C1R, ippiWTInvInitAlloc_32f_C1R, ippiWTInvGetBufSize_C1R	
iplWtFree	ippiWTFwdFree_32f_C1R ippiWTInvFree_32f_C1R	
iplWtDecompose (8U)	ippiConvert_8u32f_C1R ippiWTFwd_32f_C1R	1.2
iplWtReconstruct (8U)	ippiWTInv_32f_C1R ippiConvert_32f8u_C1R	0.8

iplXor (8U, 8S, 16U, 16S, 32S)	ippiXor	1.3
iplXorS (8U, 8S, 16U, 16S, 32S)	ippiXorC	1.2
iplXYZ2RGB (8U, 16U, 32F) pixel	ippiXYZToRGB_C3R(AC4R) ippiSwapChannels	1.1
iplXYZ2RGB (8U, 16U, 32F) plane	ippiCopy_P3C3R ippiXYZToRGB_C3R ippiCopy_C3P3R	
iplYCC2RGB (8U, 16U, 32F) pixel	ippiYCCToRGB_C3R(AC4R) ippiSwapChannels	1.8
iplYCC2RGB (8U, 16U, 32F) plane	ippiCopy_P3C3R ippiYCCToRGB_C3R ippiCopy_C3P3R	
iplYCrCb2RGB (8U) pixel	ippiSwapChannels ippiYCbCrToRGB_8u_C3R(AC4R)	2.6
iplYCrCb2RGB (8U) plane	ippiYCbCrToRGB_8u_P3R	
iplYUV2RGB (8U) pixel	ippiYUVToRGB_8u_C3R(AC4R) ippiSwapChannels	3.1
iplYUV2RGB (8U) plane	ippiYUVToRGB_8u_P3R	
iplZoom (8U, 16U, 32F)	IppiResize	1.0

Examples

Several examples of IPL function implementation are given below in pseudocode to show how Intel IPP primitives are used.

You can find the complete code of these functions as well as of all other implemented IPL functions in the source files of Intel IPP implementation of the IPL library.

iplClose

```
Image with border creation
ippiCopy
for( i=0; i<nIteration; i++ ) {
   ippiDilate3x3_8u;
}
for( i=0; i<nIteration; i++ ) {
   ippiErode3x3_8u;
}
```

iplOr(srcA, srcB, dst) planar

```
// compute intersect roi for the ippi functions
IppiSize roi = min(srcA, maskA, srcB, maskB, dst, maskD);
// set pointer to the actual data
CalcPointers(&pSrcA, &pSrcB, &pDst);
// no masks
   // operation on every plane
   for( every plane ) {
      ippiOr_<mod>_C1R(pSrcA, srcA->widthStep,
                       pSrcB, srcB->widthStep,
                       pDst, dst->widthStep, roi);
      // update the pointers pSrcA, pSrcB, pDst
      pointers += planeSizes;
   }
// with mask
   // create common mask
   ippiSet_8u_C1R(255, mask);
   ippiAnd_8u_C1IR(every mask, mask);
   // operation on every plane writes result to tmp buffer
```

```
for( every plane ) {
   ippiOr_<mod>_C1R(…, pBuf, step, roi);
   ippiCopy_<mod>_C1MR(pBuf,step,pDst,dst->widthStep,roi,mask,maskStep);
   // update pointers pSrcA, pSrcB, pDst
   pointers += planeSizes
}
```

Borders

You should pay special attention when working with functions that require border pixels, for example, filtering functions. In IPL, you can define the type of the required border, that is, determine the values of pixels that are not present in the image but are necessary for processing the image, in particular, the image borders.

Intel IPP provides an alternative processing mechanism. Intel IPP functions operate only on pixels that are part of the image. Therefore you should set a region of interest (ROI) inside the image in such a way that all neighborhood pixels necessary for processing the ROI edges can be available. In other words, the width of the remaining border outside ROI should be not less than half of the filter kernel size with the centered anchor point. The following example illustrates how you can work with the ROI.

```
void useroi() {
   int W=5,H=5,half=1,step;
   IppiSize imgSize={W,H},roiSize={W-half*2,H-half*2};
   Ipp8u*img=ippiMalloc_8u_C1(W,H,&step);
   ippiSet_8u_C1R(5,img,step,imgSize);
   img[step+1]=0;
   ippiErode3x3_8u_C1IR(img+half*step+half,step,roiSize);
   ippiFree(img);
}
```

Use the function `ippiSet()` to create a test image with a hole on the ROI border.

```
5 5 5 5 5
5 0 5 5 5
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
```

Use the function `ippiErode()`. The pointer to the data shifts to the beginning of ROI. The size of the ROI is set smaller than the image size. The function `ippiErode()` should expand (erode) the hole by one pixel in every direction. However, the hole expansion takes place only within ROI, that is, in the direction towards the center of the image. Image border pixels remain unchanged since processing region is limited by ROI and only available pixels are processed. No assumptions are made as to the pixels outside the image limits.

Processing yields a new image of a smaller size than the original image.

```
5 5 5 5 5
5 0 0 5 5
5 0 0 5 5
5 5 5 5 5
5 5 5 5 5
```

Some may find that working with “real” (existing) pixels only is not very convenient and/or acceptable. In this case you may use either a special function `ippiCopyConstBorder` to expand the original image and pad the borders with constant value pixels, or the function `ippiCopyReplicateBorder` to expand the image and copy the border pixels. The following example illustrates how the whole image can be processed.

```
void usecpy() {
   int W=5,H=5,half=1,step,STEP;
   IppiSize maskSize={2*half+1,2*half+1};
   IppiSize roiSize={W-half*2,H-half*2}, imgSize={W,H};
   IppiSize IMGSize={W+half*2,H+half*2};
   Ipp8u*img=ippiMalloc_8u_C1(W,H,&step);
   Ipp8u*IMG=ippiMalloc_8u_C1(W+half*2,H+half*2,&STEP);
   ippiSet_8u_C1R(5,img,step,imgSize);
   img[step+1]=0;
   ippiCopyReplicateBorder_8u_C1R(img,step,imgSize,IMG,STEP,
      IMGSize,half,half);
   ippiErode3x3_8u_C1IR(IMG+half*STEP+half,STEP,imgSize);
   ippiCopy_8u_C1R(IMG+half*STEP+half,STEP,img,step,imgSize);
   ippiFree(IMG);
   ippiFree(img);
}
```

As the result, erosion is carried out in every direction. The image border pixels have also been eroded.

```
0 0 0 5 5
0 0 0 5 5
0 0 0 5 5
5 5 5 5 5
5 5 5 5 5
```